KTH Electrical Engineering

Pedro Lima, João Pedro Alvito

pfrdal@kth.se, jpfa@kth.se,

MOCAP, TRUCKS AND VISUALIZATION TOOL USER'S MANUAL

Stockholm 2013

Smart Mobility Lab

Automatic Control Department

Kungliga Tekniska Högskolan

IR-EE-Dummy 2000:099

Contents

List of Figures ii

1 System Overview **1**

1.1 Key Features 3

1.2 Who are we? 4

2 Getting Started with the Tamiya Trucks **5**

2.1 T-Motes set up 6

2.2 Trucks Set Up 8

3 Getting Started with QTM **11**

3.1 How to Define a truck as a 6DOF Body 12

4 Trajectory Creation With Matlab **17**

4.1 Prerequisites 17

4.1.1 Matlab Qualysis Client 17

4.1.2 Configuration File 18

4.2 Trajectory Creation 20

5 Getting Started with the LabVIEW Program **23**

5.1 Prerequisites 24

5.2 Program Structure 25

5.3 Running the LabVIEW program 28

6 Getting Started With The Visualization Tool **31**

6.1 Prerequisites 31

6.2 Running The Visualization Tool 33

6.3 Visualization Tool 35

7 Troubleshooting **39**

7.1 Trucks 39

7.2 Trajectory Creation 40

7.3 Visualization Tool 40

List of Figures

1.1 Program structure block diagram. 1
1.2 The authors: Pedro Lima on the left side and João Pedro Alvito on the right side. 4

2.1 Connection between a PC and a T-Mote (courtesy of [5]). 6
2.2 Connection between a truck and a T-Mote (courtesy of [5]). 6
2.3 `motelist` command result on a Cygwin terminal. 6
2.4 `sf` command result on a Cygwin terminal. 6
2.5 `sf` normal execution on a Cygwin terminal. 6
2.6 `sf` ”unix_error” example. 7
2.7 `sf` ”Note: write failed” example. 7
2.8 Connections between each component in a truck. 8
2.9 Detailed connections between a T-Mote and a Polulu board (courtesy of [5]). 8

3.1 ”6DOF Tracking” page in the ”Project Options” dialog. 12
3.2 ”Acquire Body” dialog. 12
3.3 The ”New Body #6” appeared on the page. 13
3.4 The ”New Body #6” with just 4 markers associated. 13
3.5 The body was now renamed to ”Truck Grey”. Also the makers were renumbered. 14
3.6 3D reconstruction to the space views by the cameras. The new body acquired appears with a new color. 14
3.7 ”Rotate body” dialog - aligning using its own points. 15
3.8 ”Rotate body” dialog - rotating the local coordinate system. 15
3.9 ”Translate body” dialog. 16
3.10 The new body is now visible with correct label and color and with the local system. 16

4.1 Downloading the qualisys MATLAB Client. 17
4.2 Qualisys MATLAB Client installation. 18
4.3 Configuration file generator software. 18
4.4 Example of a configuration file. 19
4.5 Example of a MATLAB Client communication. 20
4.6 Example of points acquired for one lane. 20
4.7 Final trajectory generated. 21

5.1 LabVIEW QTM Plugin download in the Qualisys webpage. 24
5.2 LabVIEW program structure block diagram. 25
5.3 Q6D Euler.vi modification step 1. 26
5.4 Q6D Euler.vi modification step 2. 26
5.5 Main vi front panel. 28

6.1 MATLAB Compiler Runtime (MCR) download. 32
6.2 Connection settings. 33
6.3 Start screen of the visualization tool. 33
6.4 Visualization Tool (Pedro Lima). 35
6.5 Visualization Tool (João Pedro Alvito). 35

Chapter 1

System Overview

A block diagram of the system structure can be seen in Figure 1.1.

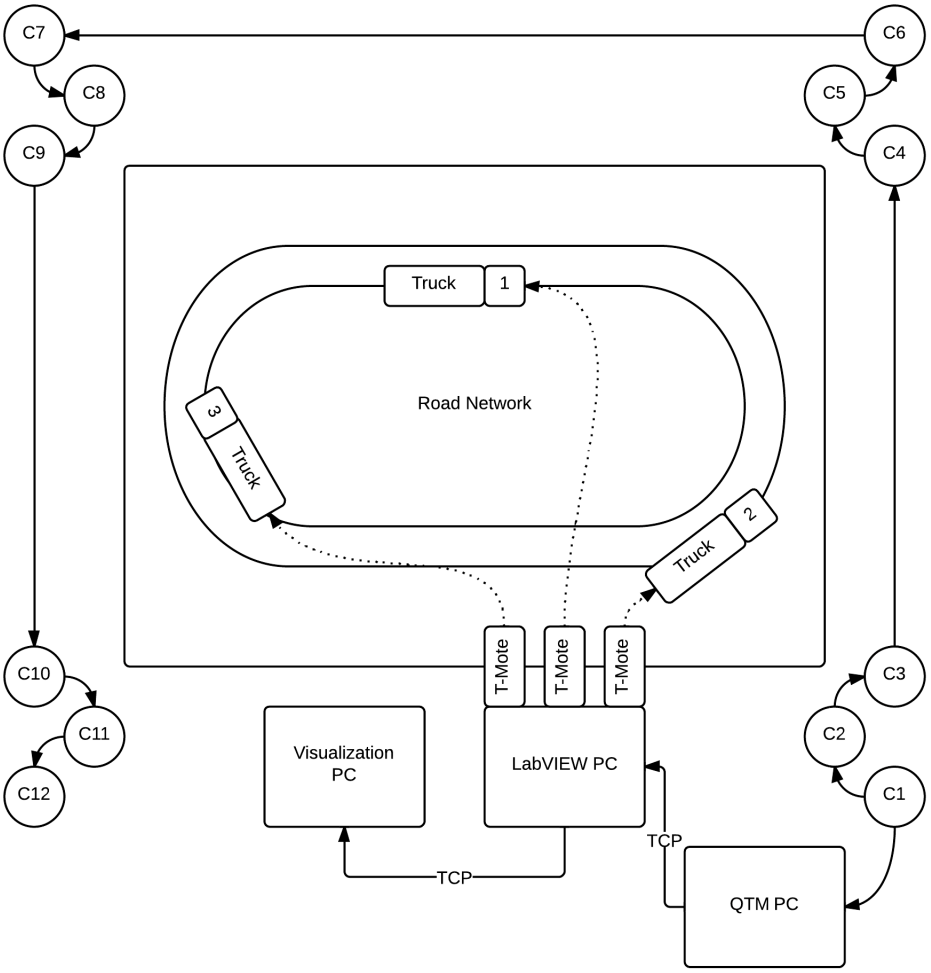


Figure 1.1: Program structure block diagram.

The overall system is supposed to represent a **real-world system** in **small scale** in which the **cameras** (C1,C2,...,C12) simulate the **GPS system**, the **LabVIEW PC** simulates an **onboard computer** responsible for the **decision making** of the truck and, naturally, the **scale trucks** represent the **real**

trucks. The **T-Motes** exist in order to make possible the communication between the PC and the trucks. Moreover, the QTM PC is the server that provides the **trucks' localization** retrieved by the cameras. Finally, the **Visualization PC** is used for **demonstration** purposes.

This system was developed in the first semester of 2013 in the **Smart Mobility Lab of the Automatic Control Department in KTH** in Stockholm under the supervision of **Jonas Mårtensson** and **Karl Henrik Johansson**. It is the **kernel** of two master's thesis, since it is the **test bed** for the experiments reported on them (see [8] and [4]).

1.1 Key Features

With the set of programs developed the user is able to:

- Use the IR markers to **create a set of waypoints** that possibly represents a trajectory, then **acquires** those points and the correspondent **interpolated trajectory** with more waypoints (see Chapter 4);
- Define the trucks as being **6DOF bodies** using the Qualysis' QTM software. This way it is possible to **track** them in **real time** with 6DOF (see Chapter 3);
- **Use the trucks** and do a wide range of operations from **platooning**, **platoon catch up** and **overtaking** to **traffic control with traffic lights** (see Chapter 5);
- Run a **visualization tool** that allows the user and the target audience to easily understand what is happening (see Chapter 6).

1.2 Who are we?

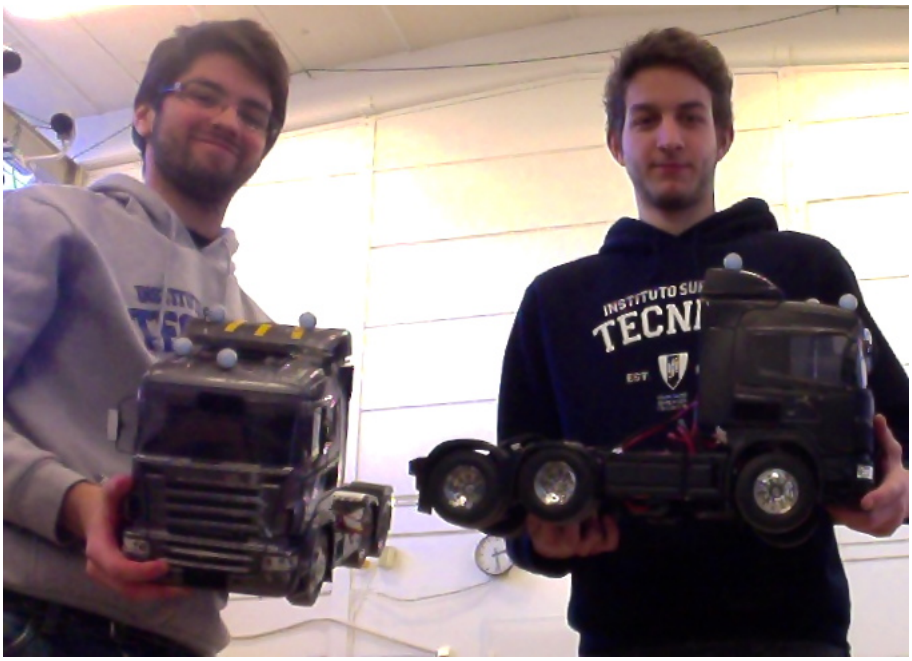


Figure 1.2: The authors: Pedro Lima on the left side and João Pedro Alvito on the right side.

We are two portuguese **master students** in **Electrical and Computers Engineering** from **Instituto Superior Técnico**, Lisbon, Portugal. We both came to KTH under an agreement between IST an KTH to take the **Master Program in Systems, Control and Robotics** (TSCRM) at KTH as part of a **Dual Degree**. This agreement includes doing our master thesis in KTH.

If you want to **know more** about this project or if you have any **doubts** don't hesitate to **contact us** (`pfrdal@kth.se` and `jpfa@kth.se`).

Chapter 2

Getting Started with the Tamiya Trucks

The **Tamiya Scania trucks** are 1/14 **scale trucks** of the real Scania V8. They are naturally one of the most important parts of the overall system.

This chapter provides a **basic tutorial of how to connect and start the trucks** and some **troubleshooting**. For a more detailed description of the material involved please refer to [5].

2.1 T-Motes set up

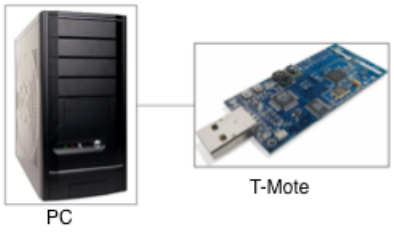


Figure 2.1: Connection between a PC and a T-Mote (courtesy of [5]).

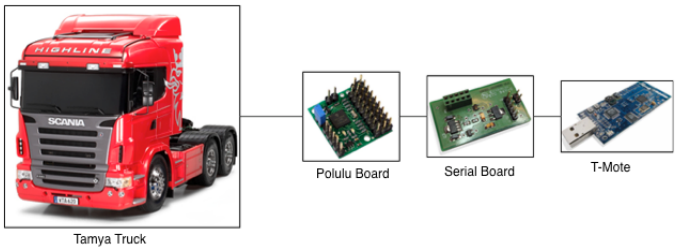


Figure 2.2: Connection between a truck and a T-Mote (courtesy of [5]).

In order to communicate with the trucks from a PC you need to know how to use the T-Motes. In the project we use the **serial-forward tool** to send data from LabVIEW to the mote attached to the PC, and the **Tiny OS** to program them. In order to learn how to install the Tiny OS and serial-forward tool please refer to [7].

In **Windows**, after having the **Cygwin** and the **serial-forward tool** installed, you should be able to see the T-Motes connected to the PC (like in Figure 2.3) using the `motelist` command.

```
el2421_user@KTH-3351 ~
$ motelist
Reference  CommPort   Description
---------- ---------- ---------------------------------------
MFT4LX0A   COM9       Port_#0006.Hub_#0005
MFSHW9KC   COM12      Port_#0003.Hub_#0006
M4AP7GUX   COM13      Port_#0004.Hub_#0006
M4AP7GUT   COM19      Port_#0004.Hub_#0005
```

Figure 2.3: `motelist` command result on a Cygwin terminal.

The T-Motes usually have its **reference number** (for example: MTF4LX0A) printed on it. If they don't, please connect **one by one** to know which COM port corresponds to each them. Then, it is possible to start a serial forward of a specific T-Mote in a specific **TCP/IP Port** like in Figure 2.4. The port chosen has to be a suitable port that is not being used by any other device. The most common ports used when serial forwarding are 9002, 9003, 9004... When trying to communicate, if everything is **okay**, you should start seeing something similar with Figure 2.5.

```
el2421_user@KTH-3351 ~
$ sf 9003 com9 115200
Warning: you're attempting to open a Windows rather that a Cygwin device.  Retry
ing with /dev/ttyS8
```

Figure 2.4: `sf` command result on a Cygwin terminal.

```
clients 0, read 0, wrote 5889
clients 1, read 0, wrote 5889
clients 0, read 0, wrote 5977
clients 1, read 0, wrote 5977
clients 0, read 0, wrote 5978
clients 1, read 0, wrote 5978
```

Figure 2.5: `sf` normal execution on a Cygwin terminal.

It is very important that you pay a lot of attention when doing serial-forward of a T-Mote in order to be able to communicate correctly with a truck. A T-Mote connected to the PC has a unique pair connected to a truck. The T-Motes used so far for that purpose are all labeled with "<COLOR> PC <ID>" and "TRUCK <COLOR> <ID>", where <COLOR> is the **color of the truck** and the <ID> is the **communication ID**[1] of the truck. The **communication ID has to be inputed in the LabVIEW program** (see Chapter 5).

If you receive messages like "Note: write failed" (see Figure 2.7) or "unix_error" (see Figure 2.6), try to kill the serial forward with CTRL+C inside of the Cygwin terminal and restart the serial forward again. If it still doesn't work, try to disconnect and connect the corresponding T-Mote and restart the serial forward. Restart the PC if nothing else works. **Always check if the IDs and COM ports you inserted in the LabVIEW program correspond to the T-Motes you are using.**

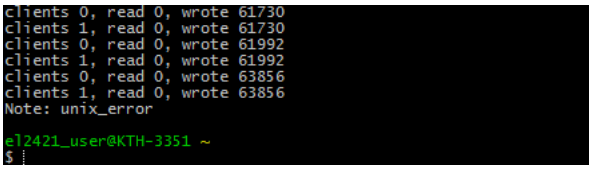

clients 0, read 0, wrote 61730
clients 1, read 0, wrote 61730
clients 0, read 0, wrote 61992
clients 1, read 0, wrote 61992
clients 0, read 0, wrote 63856
clients 1, read 0, wrote 63856
Note: unix_error
el2421_user@KTH-3351 ~
$

Figure 2.6: `sf` "unix_error" example.

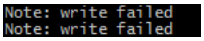


Figure 2.7: `sf` "Note: write failed" example.

If everything is okay you should see the **T-Motes' leds blinking stressfully** when you try to transmit something through them.

[1] It is not the same thing as the radio channel number!

2.2 Trucks Set Up

The **truck connections** are exemplified in figure 2.8 and 2.9.

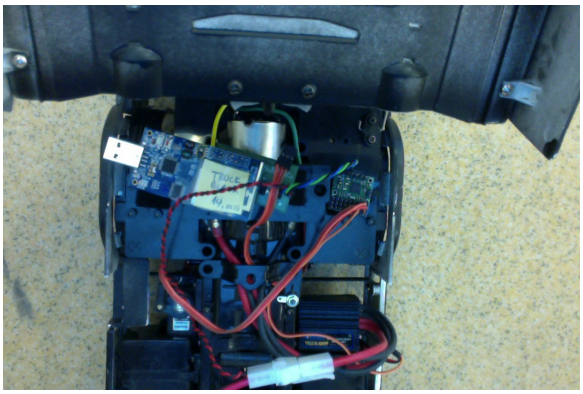


Figure 2.8: Connections between each component in a truck.

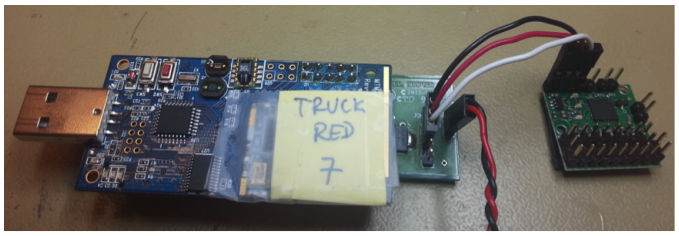


Figure 2.9: Detailed connections between a T-Mote and a Polulu board (courtesy of [5]).

The following connections have to be made on the truck:

- Connect the **2-pin female jumper wire** of the **power cable** to the **serial board**. **Pay attention to the polarity of the board**;
- Connect the **2-pin female-female jumper** between the **serial board** and the **Polulu board**. **Pay attention to the polarity of the board**;
- Connect the **3 wire cable** of the **speed servo** to the **Polulu board**. **Pay attention to the cable order**;
- Connect the **3 wire cable** of the **steering servo** to the **Polulu board**. **Pay attention to the cable order**;
- Connect the **T-Mote** to the **serial board** so that the **rightmost 10 pins** of the T-Mote are connected;
- Set the **motor switch** to **"On"**;
- Connect the **battery female connection plug** to the **male connection plug of the power cable**.
- Connect the **power cable female connection plug** to the **speed servo male connection plug**.

When everything is connected the **yellow LED** should turn on on the **Polulu board**. If you see a **red LED** this can mean that the **batteries are low**, that you **misconnected some of the cables** or the **T-Mote is misconnected to the serial board**.

When you try to communicate with the T-Mote that is on the truck you should see a **blinking blue LED on the T-Mote** and a **blinking green LED on the Polulu board**. If you don't see anything happening, **check if the IDs and COM ports you inserted in the LabVIEW program correspond to the T-Motes you are using.**

To charge the batteries you should connect the battery to a battery charger.[2] The charging rate

[2]The one used in the Smart Mobility Lab is the Vector AC/DC NX85 Variable Output Charger.

must be $0.5A$ and it takes about 10 hours for the battery to be fully charged. For safety reasons, **it is important to never forget a charging battery while no one is around**.

Chapter 3

Getting Started with QTM

Qualisys Track Manager (QTM) is a Windows-based data acquisition software with an interface that allows the user to perform 2D and 3D motion capture.

This chapter provides a **basic tutoria**l of how to use QTM in order to **define a truck as a 6DOF rigid body**. For a more detailed description of the software please refer to [1] and [3][1].

[1] Available on the Qualysis website: `http://www.qualisys.com`. It requires log in credentials.

3.1 How to Define a truck as a 6DOF Body

The trucks to be tracked are equipped with **markers**. These are small IR-reflective spheres placed on the top of the truck. A **minimum of 3 markers** is required in order to define a **6DOF Rigid Body**. We placed 4 markers per truck so that the trucks are more robustly tracked. Each truck has its unique marker configuration in order to be unequivocally recognized.

In order to define a 6DOF body you can follow the **next steps**.

- **Step 1**: Open the **"6DOF Tracking" page** that can be found in the **"Project Options" dialog**. There you will see the already defined 6DOF Rigid Bodies, define new ones or load old ones. To acquire a new body, **right-click "Acquire Body"** on the right side of the dialog.

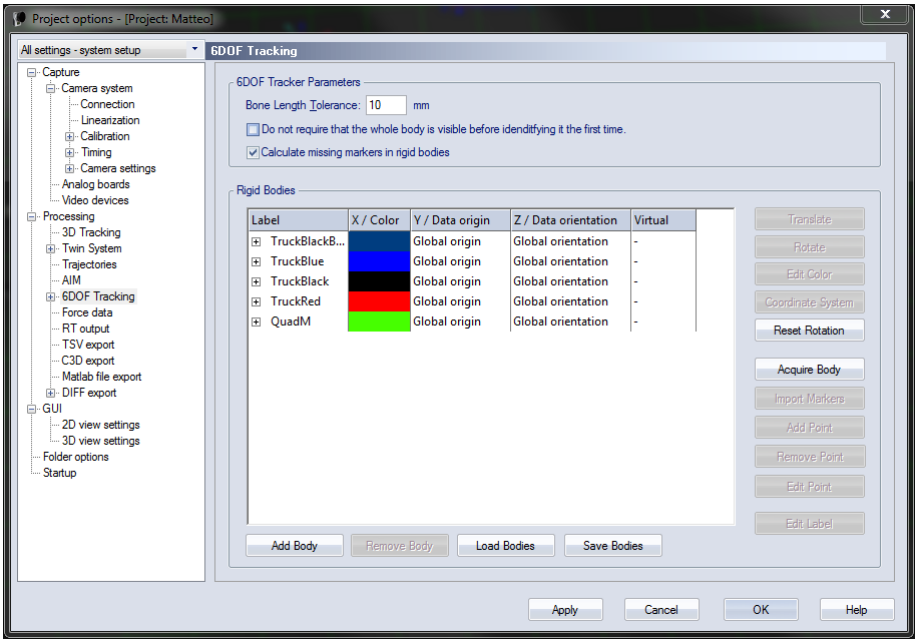


Figure 3.1: "6DOF Tracking" page in the "Project Options" dialog.

- **Step 2**: Another dialog should appear. Then click **"Start"**.

Figure 3.2: "Acquire Body" dialog.

- **Step 3**: All the markers that can be seen by the cameras will appear under the "New Body" markers list. You should **delete from the list all the acquired markers that do not make part of the body you want to define**. The best way to do this is either by only putting the body you are defining on the visible are or knowing more or less the coordinates of the markers which make part of the new body and you deleting all the others.

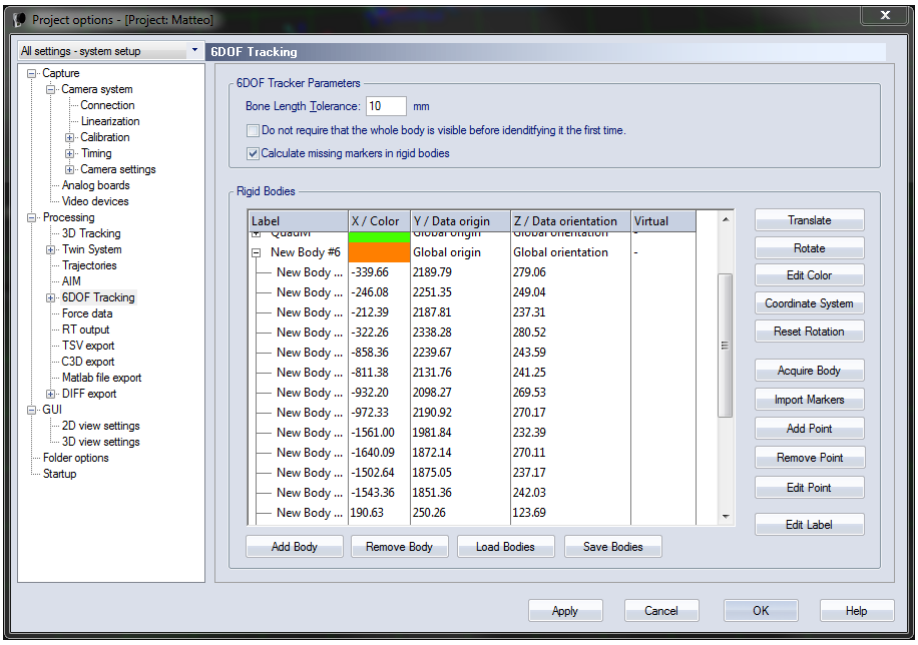


Figure 3.3: The ”New Body #6” appeared on the page.

- **Step 4**: After deleting all the markers from the list, **you should have just 4 markers** in order to define a truck as a 6DOF Rigid Body. Now, you should **rename** the ”New Body” to a logical name like ”Truck Grey” and you **must renumber** all the markers acquired from 1 to N where N is the number of markers of the new Rigid Body you want to define. If you want you can also **change the color** that the Rigid Body will assume when recognized.

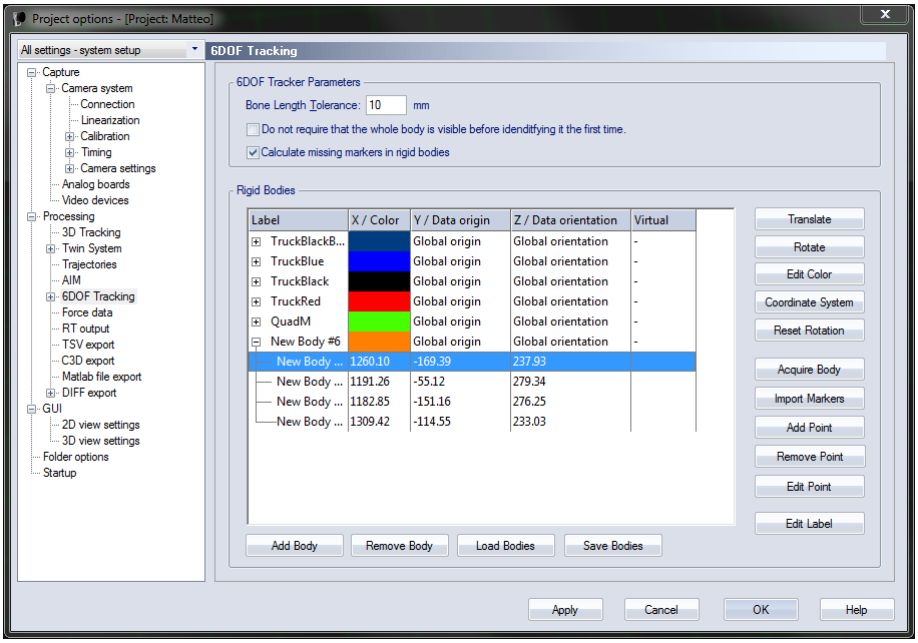


Figure 3.4: The ”New Body #6” with just 4 markers associated.

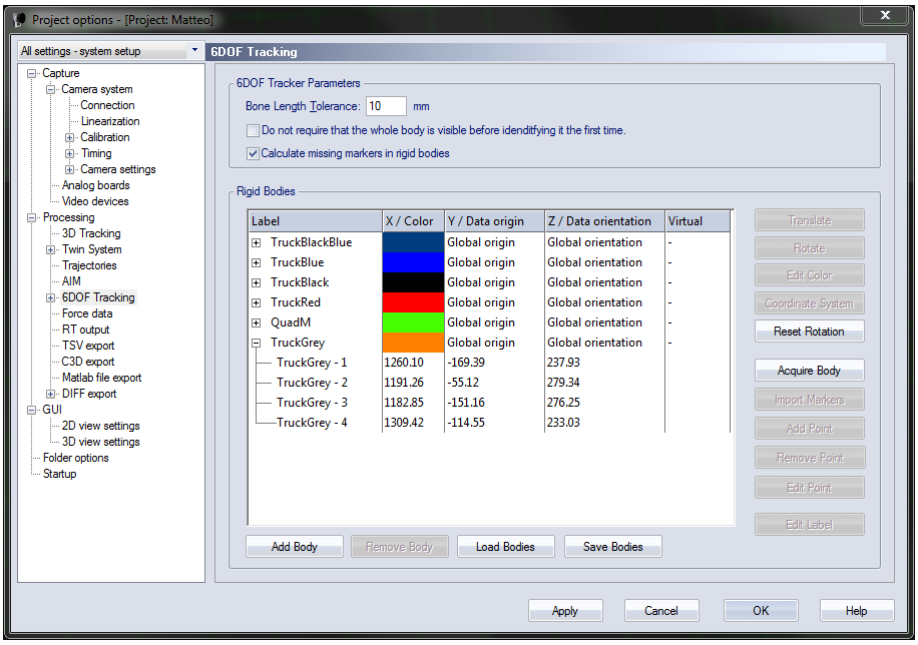


Figure 3.5: The body was now renamed to ”Truck Grey”. Also the makers were renumbered.

- **Step 5**: You should now see the new body **recognized** with the color you have chosen. In Figure 3.6, this corresponds to the 4 grey points next to the X axis. The next step is to **associate a local coordinate system to the Rigid Body**.

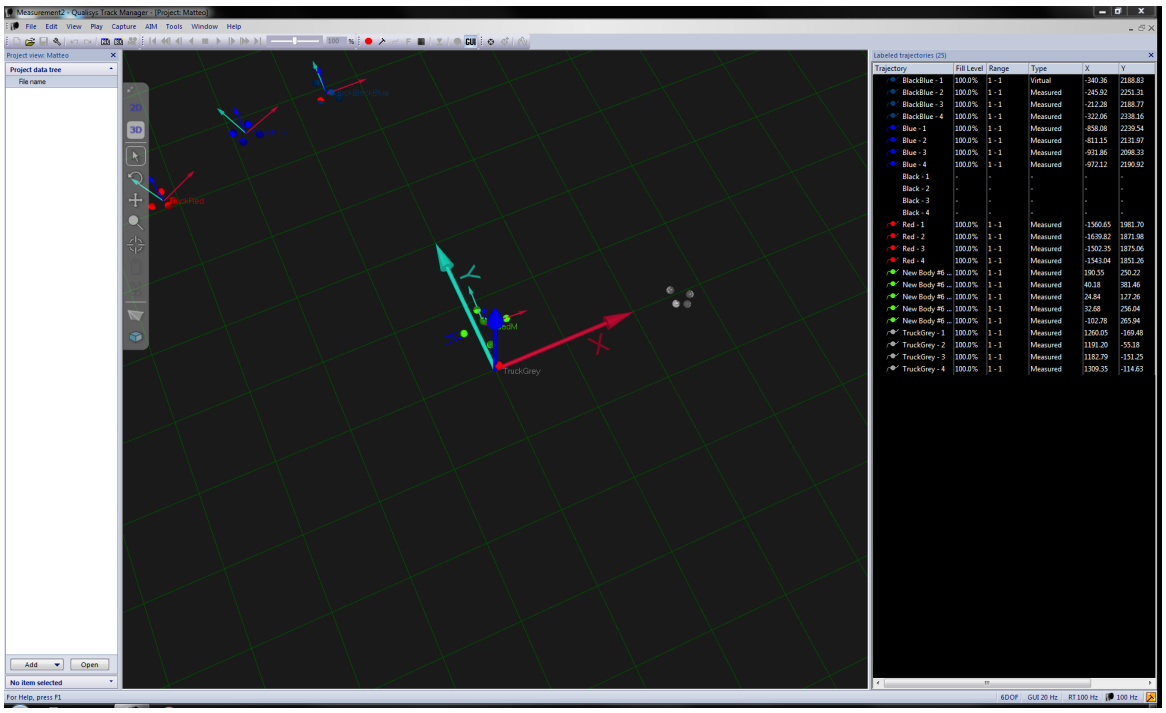


Figure 3.6: 3D reconstruction to the space views by the cameras. The new body acquired appears with a new color.

- **Step 6**: To **associate the X and Y axis of local coordinate system to certain markers**, right-

click **"Translate"** on the right side of the "6DOF Tracking" page in the "Project Options" dialog. A new dialog box should appear. Then select **"Align the body using its points"** and **input the numbers** of the markers you want to be recognized as the X axis and the Y axis of the body.

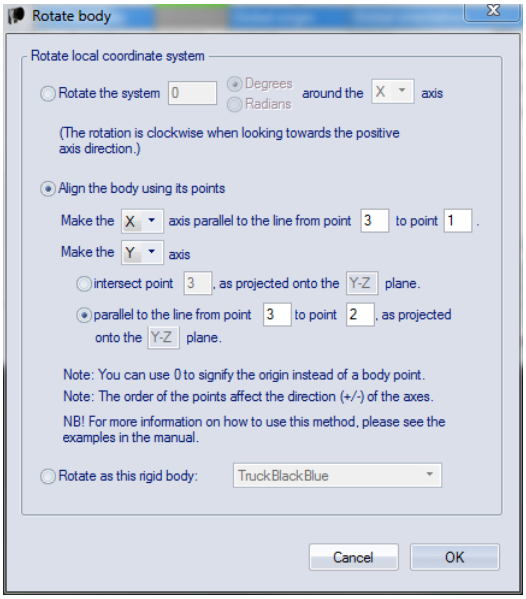

Rotate body
Rotate local coordinate system
Rotate the system 0 Degrees Radians around the X axis
(The rotation is clockwise when looking towards the positive axis direction.)
Align the body using its points
Make the X axis parallel to the line from point 3 to point 1 .
Make the Y axis
intersect point 3 , as projected onto the Y-Z plane.
parallel to the line from point 3 to point 2 , as projected onto the Y-Z plane.
Note: You can use 0 to signify the origin instead of a body point.
Note: The order of the points affect the direction (+/-) of the axes.
NB! For more information on how to use this method, please see the examples in the manual.
Rotate as this rigid body: TruckBlackBlue
Cancel
OK

Figure 3.7: "Rotate body" dialog - aligning using its own points.

- **Step 7**: Some final adjustments like **rotating the coordinate system around a certain axis** can be done. On the same **"Rotate"** dialog, **if acquiring one of our marker configurations**, you should rotate more or less -20^o degrees around the Y axis in order to align the local coordinate system with the world coordinate system.

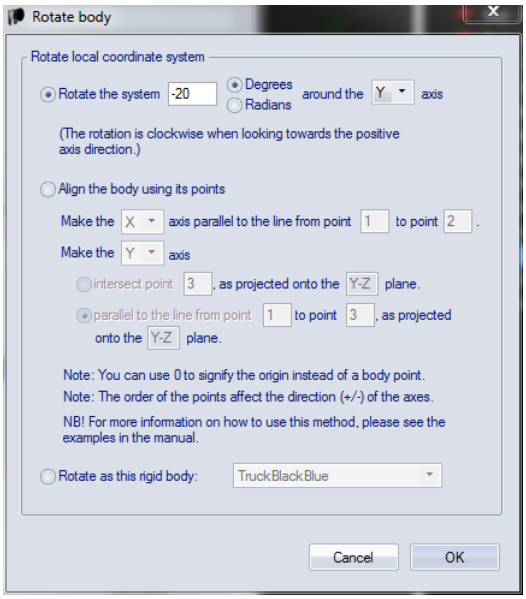

Rotate body
Rotate local coordinate system
Rotate the system -20 Degrees Radians around the Y axis
(The rotation is clockwise when looking towards the positive axis direction.)
Align the body using its points
Make the X axis parallel to the line from point 1 to point 2 .
Make the Y axis
intersect point 3 , as projected onto the Y-Z plane.
parallel to the line from point 1 to point 3 , as projected onto the Y-Z plane.
Note: You can use 0 to signify the origin instead of a body point.
Note: The order of the points affect the direction (+/-) of the axes.
NB! For more information on how to use this method, please see the examples in the manual.
Rotate as this rigid body: TruckBlackBlue
Cancel
OK

Figure 3.8: "Rotate body" dialog - rotating the local coordinate system.

- **Step 8**: To **translate the coordinate system to the geometric center of the body**, right-click

”Translate” on the right side of the dialog. A new dialog box should appear. Then select **”To the geometric center of the body (the average of the body points)”**.

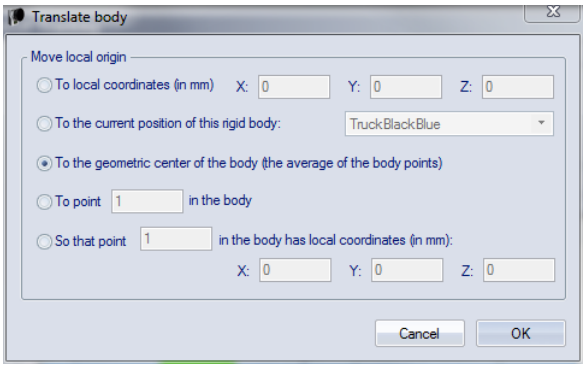


Figure 3.9: ”Translate body” dialog.

- **Step 9**: The new body should now appear with the **correct label and color and with the local coordinate system**.

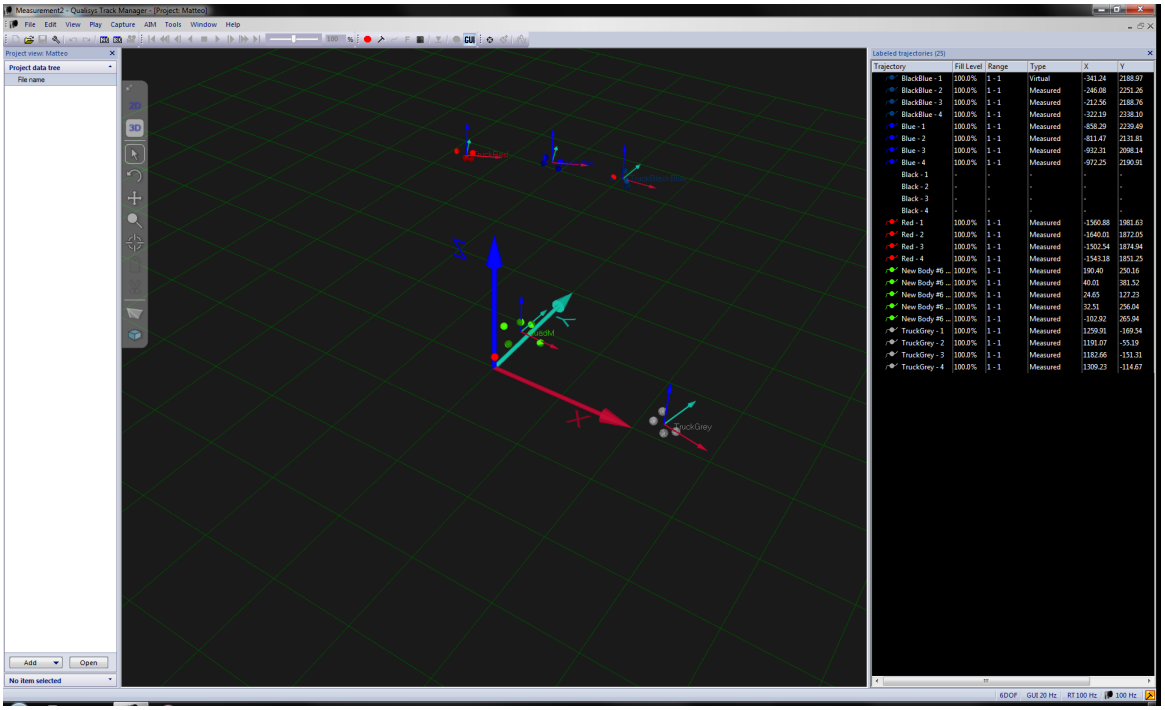


Figure 3.10: The new body is now visible with correct label and color and with the local system.

If you are not getting the desired result please make sure you followed correctly all the steps and refer to [1] and [3] for more detailed information.

Chapter 4

Trajectory Creation With Matlab

In order to move the trucks along the road network we have to create the trajectories. They are composed of a sequence of points that will be followed by the trucks. There are several possible ways to create the trajectory. Here we are going to present the method that we used and all the steps you should take if you want to replicate it in some way.

4.1 Prerequisites

4.1.1 Matlab Qualysis Client

To communicate with the **Qualysis** system you need the **MATLAB Client**(which you can download in the Qualysis website[1]). It will require login credentials, but if you use a computer in the **Smart Mobility Lab** you will have the client already installed. Naturally, to use the **MATLAB Client** you need to have installed in your computer a valid **MATLAB** license.

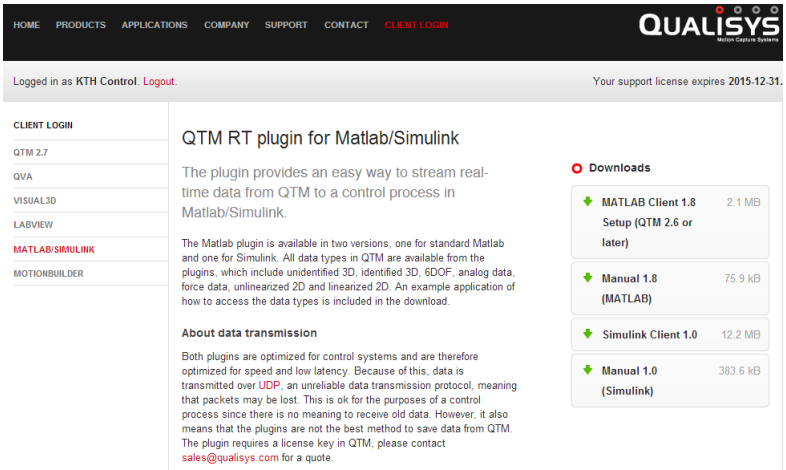

HOME
PRODUCTS
APPLICATIONS
COMPANY
SUPPORT
CONTACT
CLIENT LOGIN
QUALISYS
Logged in as KTH Control. Logout.
Your support license expires 2015.12.31.
CLIENT LOGIN
QTM 2.7
QVA
VISUAL3D
LABVIEW
MATLAB/SIMULINK
MOTIONBUILDER
QTM RT plugin for Matlab/Simulink
The plugin provides an easy way to stream real-time data from QTM to a control process in Matlab/Simulink.
The Matlab plugin is available in two versions, one for standard Matlab and one for Simulink. All data types in QTM are available from the plugins, which include unidentified 3D, identified 3D, 6DOF, analog data, force data, unlinearized 2D and linearized 2D. An example application of how to access the data types is included in the download.
About data transmission
Both plugins are optimized for control systems and are therefore optimized for speed and low latency. Because of this, data is transmitted over UDP, an unreliable data transmission protocol, meaning that packets may be lost. This is ok for the purposes of a control process since there is no meaning to receive old data. However, it also means that the plugins are not the best method to save data from QTM. The plugin requires a license key in QTM, please contact sales@qualisys.com for a quote.
Downloads
MATLAB Client 1.8 Setup (QTM 2.6 or later) 2.1 MB
Manual 1.8 (MATLAB) 75.9 kB
Simulink Client 1.0 12.2 MB
Manual 1.0 (Simulink) 383.6 kB

Figure 4.1: Downloading the qualisys MATLAB Client.

[1] www.qualysis.com

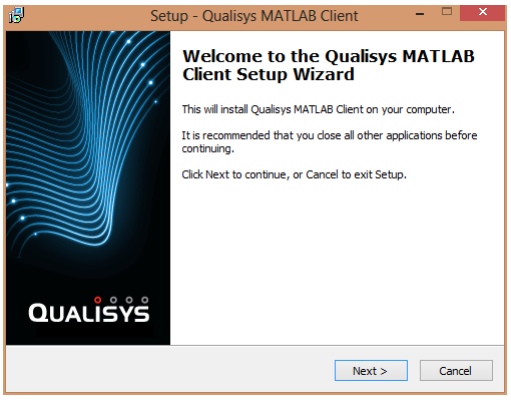

Setup - Qualisys MATLAB Client
Welcome to the Qualisys MATLAB Client Setup Wizard
This will install Qualisys MATLAB Client on your computer.
It is recommended that you close all other applications before continuing.
Click Next to continue, or Cancel to exit Setup.
QUALISYS
Next >
Cancel

Figure 4.2: Qualisys MATLAB Client installation.

If you need to use it in a different computer just download the installation file and follow the normal procedures to have the client installed. Note that a computer with Microsoft Windows installed is required.

4.1.2 Configuration File

When communicating with the Qualysis system, the MATLAB Client uses a configuration file where all the settings are defined. With the purpose of creating a more dynamical experience, we created a MATLAB application that generates the configuration file according to the settings chosen by the user.

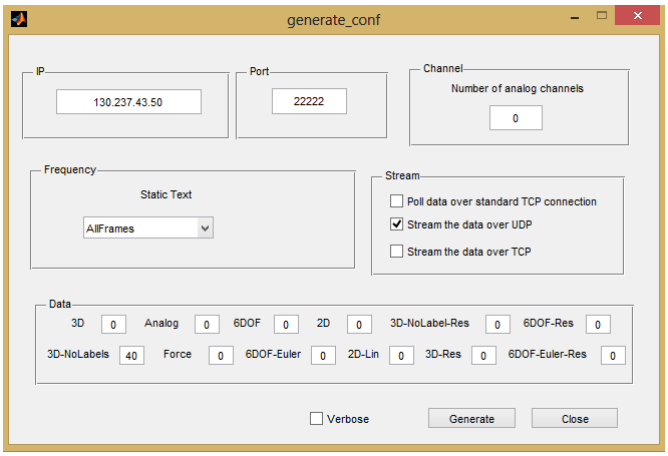


Figure 4.3: Configuration file generator software.

To know more about each parameter of the configuration file we recommend you to refer to [2]. In Figure 4.4, as an example, we present a sample of a generated configuration file with the settings shown in Figure 4.3.

QMC_conf.txt

```
# IP address of the QTM_RT server
<IP:130.237.43.50>

# Port used
<PORT:22222>

# Frequency to fetch the data from the QTM RT server. This is only used when
# streaming data (STREAM set to 1), se below.
# <FREQ:FrequencyDivisor:n>   # The camera frequency divided by n.
# <FREQ:Frequency:n>          # Stream data in n Hz
# <FREQ:AllFrames>            # Stream data with camera frequency.
<FREQ:AllFrames>

# Stream data
# 0 : Request (poll) data over standard TCP connection.
# 1 : Stream the data over UDP.
# 2 : Stream the data over TCP.
<STREAM:0>

# Verbose
# 0 : Do not print frame number and timestamp.
# 1 : Print frame number and timestamp.
<VERBOSE:0>

# Output data size to MATLAB. Amount of objects sent to output for each
# component. Enter a value for each component. Use 0 to disable a component.
<3D:0>                    # Max number of markers to receive.
<3D-NoLabels:40>           # Max number of unlabeled markers to receive.
<AnalogSingle:0>          # Max number of analog devices to receive data from.
<ForceSingle:0>           # Max number of forces to receive.
<6DOF:0>                  # Max number of 6DOF bodies to receive.
<6DOF-Euler:0>            # Max number of 6DOF-Euler bodies to receive.
<2D:0>                    # Max number of 2D points to receive from one camera.
<2D-Lin:0>                # Max number of 2D-Lin points to receive from one camera.
<3D-Residual:0>           # Max number of markers with residual to receive.
<3D-NoLabels-Residual:0> # Max number of unlabeled markers with residual to receive.
<6DOF-Residual:0>         # Max number of 6DOF bodies to receive.
<6DOF-Euler-Residual:0>  # Max number of 6DOF-Euler bodies to receive.

# Max number of analog channels to receive data from.
<CHANNEL:0>
```

Figure 4.4: Example of a configuration file.

4.2 Trajectory Creation

Now that you have everything needed to acquire a trajectory, start by running the file *trajectory.m* that will start the acquisition process.

In case you already have a configuration file in the same folder you are running the *trajectory.m* file, the program will use that file; otherwise, the configuration file generator will start and allow you to create a new one. This file will be used as the configuration file to the acquisition process. If the communication is started successfully, it should appear something similar to Figure 4.5.

```
>> trajectory
Qualisys MATLAB client v1.8
Connecting to : 130.237.43.50 on port 22222
Active data:  3DnoLabels
```

Figure 4.5: Example of a MATLAB Client communication.

After that, something similar to Figure 4.6 should appear.

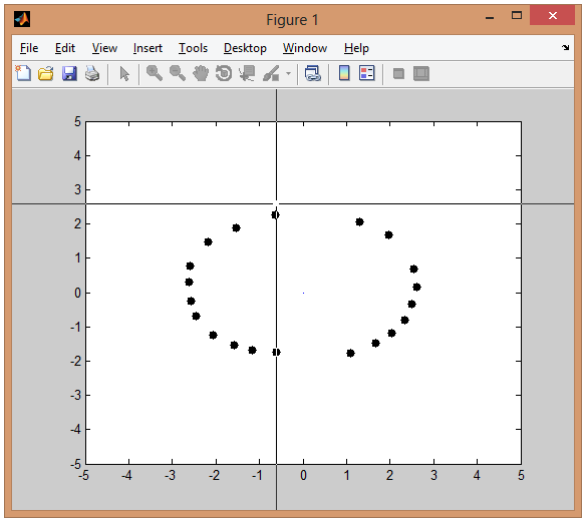


Figure 4.6: Example of points acquired for one lane.

Here you are asked to choose the first two points of the trajectory. This is done with the function `ginput` of **MATLAB**, so to choose the points you just nee to click twice (once for each point) somewhere close to the desired points. We need to choose **two points** because at this point we are stipulating the direction of the lane. Since the **LabVIEW** program makes the truck follow the trajectory sequentially, it is important to choose beforehand the direction of the trajectory.

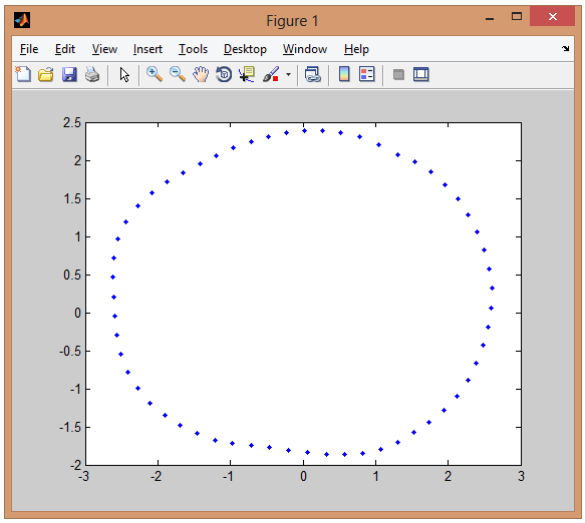


Figure 4.7: Final trajectory generated.

We use an **interpolation** process to create more waypoints because this creates a **smoother trajectory** which will create a more realistic trajectory following of the trucks. After that, the result is similar to Figure 4.7.

In the end, two files are created. One is *"WPList.txt"*, a file with a list of the coordinates in meters of all the trajectory waypoints. Note that it is relevant to choose if you want numbers' floating points to be '**.**' or '**,**'. Unfortunately, different versions of **LabVIEW** require different nomenclatures.

Version	Nomenclature
LabVIEW 2011	,
LabVIEW 2012	.

Table 4.1: **LabVIEW** version floating point nomenclature.

It will also be created another file named *"TrajectoryLUT.txt"*. This is a lookup table with the distances between every waypoint to every other waypoints in meters. Once again, pay attention to the **nomenclature** that you choose. We opted to calculate the distances between every point of the trajectory in the trajectory creation stage, mainly because it only needs to be done once and since it is somewhat computational costly, it is beneficial to do it in **MATLAB** and not in **LabVIEW**.

Now that you have the trajectory files, the last step is to copy them to the correct folders in the computer that is running the **LabVIEW** program.

When creating a road network, you may need to create several roads or lanes. For that you need to do the process previously explained for every road you want to create. After that you get two files for every road, one containing the waypoints and the other one with the distances between the waypoints. If you need to create connected roads, i.e. roads that have transition waypoints between them, you just create the roads normally and then in the **LabVIEW** program you need to specify the number of those waypoints.

Chapter 5

Getting Started with the LabVIEW Program

LabVIEW is a powerful **National Instruments** software that allows the user to program using a graphical interface like blocks and connecting wires. The developed LabVIEW program achieves the following objectives:

- Creates the **connection** between the **PC** and the **trucks** using the serial forwarded T-Motes;
- Creates the **TCP connection** between the PC and another PC running the **Visualization Tool**;
- Fetches the **MoCap** data;
- **Controls** the trucks behavior.

5.1 Prerequisites

We recommend that the program is used in a PC running **Windows 7 or higher** with **LabVIEW 2011 or higher**. Note that only the **LabVIEW 2012** is compatible with **Windows 8**. Although there are compatible versions of LabVIEW for other platforms like Linux and Mac OSX it is not guaranteed that everything works like in Windows. Problems like doing the T-Motes serial forwarding and fetching the MoCap data can arise. LabVIEW can be **downloaded** from the National Instruments website[1] and, for instance, a **student license** can be downloaded from **KTH Prodgist** website[2] and then activated. In order to do the serial forward of the T-Motes you have to install Cygwin and TinyOS (see Chapter 2). To be able to fetch the data from the **MoCap** it is essential to download the **LabVIEW QTM Plugin** from **Qualysis** website[3]-for that it is needed a client **username** and **password**. The installation is pretty straightforward and the steps are pretty common among all the Windows installations and is explained next.

In order to find the **LabVIEW QTM Plugin** you just need to go to the **Qualysis** website and you will see its download link on the **right side** of the page.

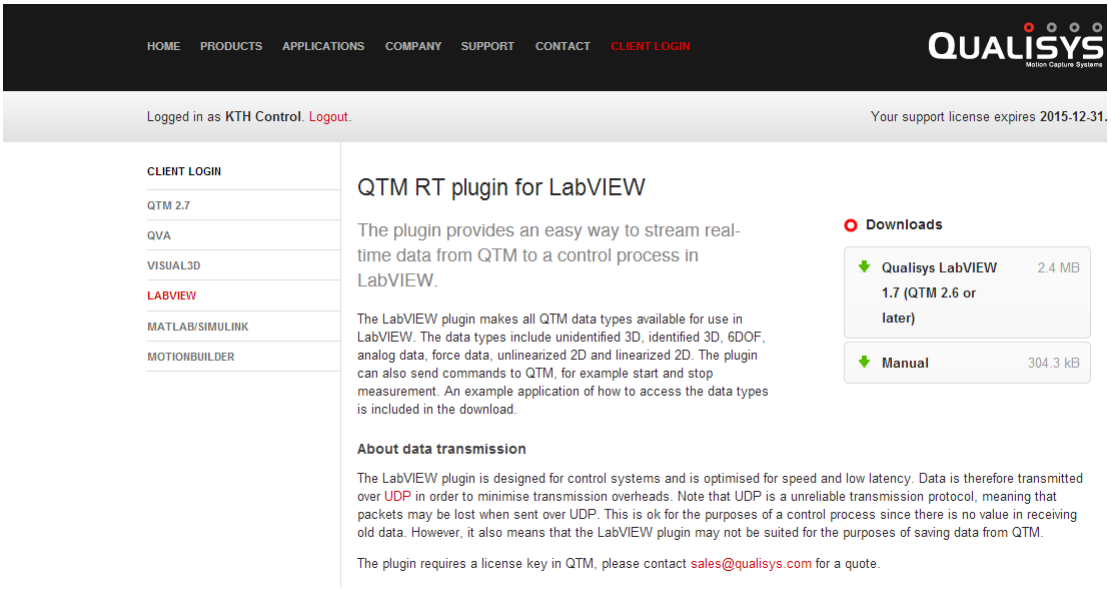

HOME
PRODUCTS
APPLICATIONS
COMPANY
SUPPORT
CONTACT
CLIENT LOGIN
QUALISYS
Logged in as KTH Control. Logout.
Your support license expires 2015-12-31.
CLIENT LOGIN
QTM 2.7
QVA
VISUAL3D
LABVIEW
MATLAB/SIMULINK
MOTIONBUILDER
QTM RT plugin for LabVIEW
The plugin provides an easy way to stream real-time data from QTM to a control process in LabVIEW.
The LabVIEW plugin makes all QTM data types available for use in LabVIEW. The data types include unidentified 3D, identified 3D, 6DOF, analog data, force data, unlinearized 2D and linearized 2D. The plugin can also send commands to QTM, for example start and stop measurement. An example application of how to access the data types is included in the download.
Downloads
Qualisys LabVIEW 1.7 (QTM 2.6 or later)
2.4 MB
Manual
304.3 kB
About data transmission
The LabVIEW plugin is designed for control systems and is optimised for speed and low latency. Data is therefore transmitted over UDP in order to minimise transmission overheads. Note that UDP is a unreliable transmission protocol, meaning that packets may be lost when sent over UDP. This is ok for the purposes of a control process since there is no value in receiving old data. However, it also means that the LabVIEW plugin may not be suited for the purposes of saving data from QTM.
The plugin requires a license key in QTM, please contact sales@qualisys.com for a quote.

Figure 5.1: LabVIEW QTM Plugin download in the Qualisys webpage.

In the folder where the LabVIEW project is in, there should be as well a folder with the name *RoadNetwork2011* and *RoadNetwork2012*, which are used inside of LabVIEW 2011 and LabVIEW 2012 respectively. Inside of those there should appear several *.txt* files that describe the **road network**. Their creation is explained in Chapter 4.

[1] www.ni.com
[2] www.kth.se/student/support/itsc/progdist/software-for-windows/labview-2012-spring-1.337984
[3] www.qualysis.com

5.2 Program Structure

A **block diagram** representing the LabVIEW **program structure** is shown on figure 5.2.

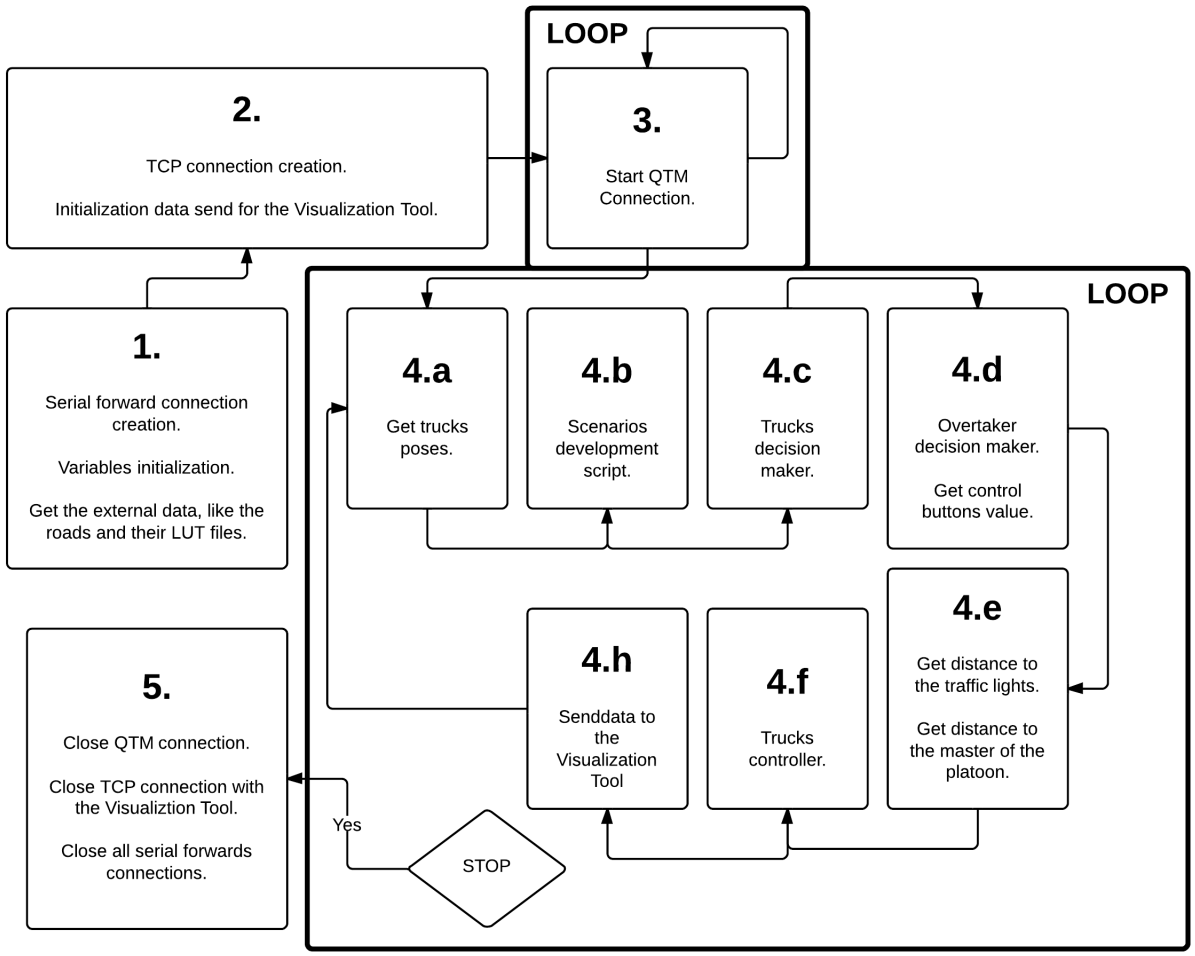


Figure 5.2: LabVIEW program structure block diagram.

- **1.** This is where the program execution **starts**. At this point you should have created the **serial forward** for each mote in the right **COM ports**. You should have checked as well if the **Road Network files** exist or not. One of the main hidden aspects about this part is the setting of the **midranges values** of the trucks. The midranges values are the values that are going to be considered as 0 in the truck. This means that if we send the value 127 both for **speed** and for **steering**, the truck should be **stopped** and with its **wheels heading straightforward**. If we sent **below** that value, the truck should go **backward** (in the case of the speed value) and **turn left** (in the case of the steering value) and **vice versa for upper values**. The setting of these midranges values is the **first value sent** to the truck **after the connection** is made;

- **2.** A **TCP connection** is created and the PC where the program is running works as a **server** for the **Visualization Tool**. The **first** message sent has the following **format**:

NRoads	Road$_1$ Length	Road$_1$ WP	...	Road$_N$ Length	Road$_N$ WP	Scenario Owner

Scenario Name	NTrucks	Bridge Status

- **3.** The **QTM connection** is done in an infinite while loop in order to provide the **latest value** possible;

- **4.a** The **trucks poses** are fetched using the method presented in a **LabVIEW demo** that is installed along with the **LabVIEW QTM Plugin**. Nevertheless, we have changed the block *Q6D Euler.vi*. The following changes were made:

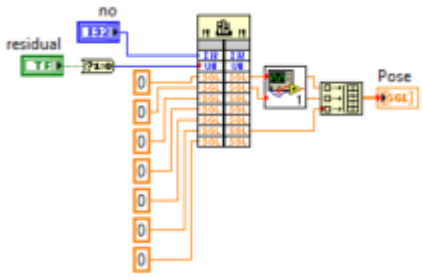


Figure 5.3: Q6D Euler.vi modification step 1.

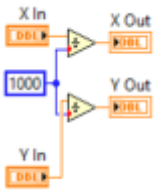


Figure 5.4: Q6D Euler.vi modification step 2.

These modifications allow the Q6D Euler.vi to output an **array** with the coordinates x and y in **meters** (instead of millimeters) and the **orientation** yaw in **degrees**. The figure 5.4 corresponds to the **subvi** block in figure 5.3;

- **4.b** Whenever a **scenario** is selected to run, a **MathScript** will **evaluate** and **change** the control variables in order to run a **user-defined scenario**;

- **4.c** The **decision maker** is the **brain** of the trucks. It decides the **trajectory waypoint** to follow, the **reference speed** changes and the **state transitions**. In this case, each state is a **road lane**. We have an **Outer Lane** (state 1), an **Inner Lane** (state 2) and an **Inner Road** (state 3). It is only allowed to transition between the states 2 and 3;

- **4.d** The **overtaking decision maker** does nothing more often than not. When the **platoon master is changed** and that truck was **already in a platoon**, the new master will **overtake** in order to go to the front of the platoon. The overtaking decision maker decides when the truck should **start its overtaking maneuver** and when it should **finish it**.

  At this point, the **buttons' values** that the user can change in the Main.vi front panel are evaluated.

- **4.e** If there is a **platoon master**, all the trucks in the **same lane** as the master will have its **distance to the master updated**. Since the **Outer Lane** (state 1) and the **Inner Lane** (state 2) have **traffic lights,** the **distance of each truck to them** is also updated. This is only applicable to the states where traffic lights exist.

- **4.f** Using the information given by the decision maker **each truck is controlled using two PID controllers**, one for the **speed** and the other for the **steering**.

- **4.h** All the updated information is sent to the **Visualization Tool** in order to be represented in real-time. The message sent has the following **format**:

STOP	Traffic Lights State	Traffic Lights Position	MasterID	Platooning Reference Distance

Real Speed$_1$	...	Real Speed$_N$	Current State$_1$	...	Current State$_N$

Current WP$_1$	...	Current WP$_N$	Distance to Traffic Lights$_1$	...	Distance to Traffic Lights$_N$

Distance to Master$_1$	...	Distance to Master$_N$	Reference Speed$_i$	...	Reference Speed$_N$

Platoon Array	Stop Array	Truck$_1$ Pose	...	Truck$_N$ Pose	Information String

- **5.** When the button **STOP** is pressed (also called the **panic button**) everything stops. The **trucks stop** and the **connections to QTM**, to the **T-Motes** and to the **Visualization Tool** are **closed**.

5.3 Running the LabVIEW program

In order to run the LabVIEW program you just need to know how to use the **Main vi**. The **Main vi front panel** can be seen in Figure 5.5 and is the **control panel** of the program. At first, one may think that it is too much information at the same time, but **everything is organized and has its own logic**. Each part of the front panel will be explained in detail.

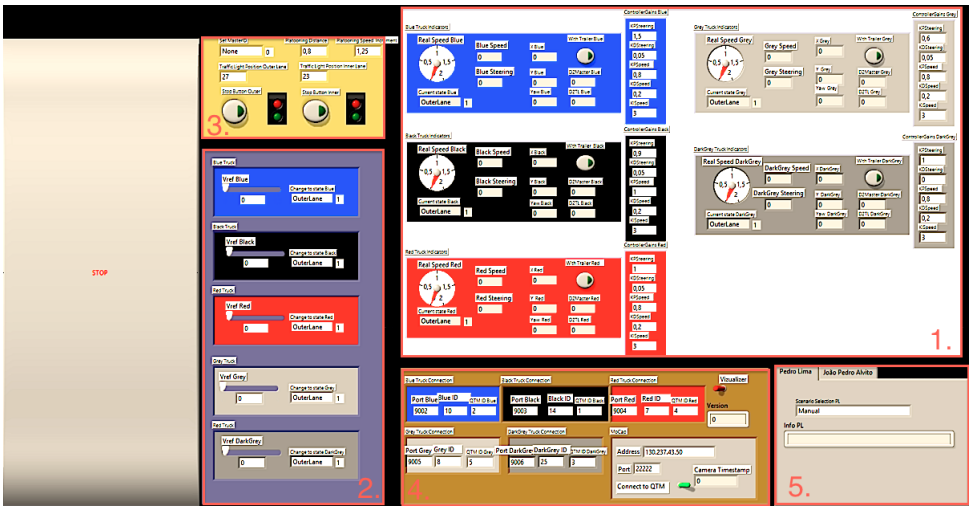


Figure 5.5: Main vi front panel.

- **1.** This panel is mostly consisted of **indicators**. There we can see, for each truck, its **Real Speed**, **Current State**, **Current Pose** (x, y and yaw), **Distance to Master**, **Distance to Traffic Lights** and the values of **Speed** and **Steering** in **bits** sent to the truck. Each truck has an associated **panel color** which, naturally, corresponds to its **real color**. In this panel, one can select if the truck has a **trailer** or not, which will influence the program's perception about the **truck's length**. In order to easily tune the controllers for each truck, the **controller gains** are available as well for each truck.

- **2.** If the program is on the **Manual Mode** (see point **5.**) you will do most of the modifications on this panel. With a sliding bar you can change the **Reference Speed** of each truck and you can **Change the State** where the truck is. Naturally, the truck will only change its state in the **transition points**, between the **Inner Road** and the **Inner Lane**. **If the program isn't on the Manual Mode you don't have to worry about this**.

- **3.** Once again, if the program is on the **Manual Mode** (see point **5.**) you can modify several parameters of the system. These parameters are: the **Reference Gap Distance** between trucks when they are on a platoon formation, the **Speed Increment** ($V_{ref} = SpeedIncrement \times V_{Master}$ when the truck starts catching up, the **Traffic Lights Position** in Outer and Inner Lanes and each **Traffic Light Status**. The position of the traffic lights can be hard to understand and to set. The values that are now set by default are **waypoints** of the corresponding lane. In order to change these values you have to do trial and error until you find the position wanted.

- **4.** The **missusage** of the control boxes on this panel can lead to a **strange behavior** of the

program. For each truck you can set the corresponding **ID** used in the T-Motes communication, the **Port** used in the T-Motes serial forwarding (see Chapter 2) and **QTM ID**, which is the label of that truck on the MoCap (see Chapter 3). It's also possible to see the **Version** of the LabVIEW where the program is running on and set the **Visualization Tool** "**On**" and "**Off**". If this button is set to "**On**", you **must run the Visualization Tool** (see how to do it in Chapter 6), otherwise **the LabVIEW won't run**. The program **will not run without having the values from the MoCap**, so if the connection is okay, you will see the **Camera Timestamp** increasing. Always check the **QTM computer IP address** if you notice that the **Camera Timestamp isn't changing**.

- **5.** In this panel you can choose two tabs. The "**Pedro Lima**" **tab** and the "**Joao Pedro Alvito**" **tab**. Those are the names of the authors of the program. The tabs exist in order to **run different scenarios** created by those using the program. In both tabs you have the option "**Manual**" where you can control the trucks using the panels explained above, otherwise you can choose a scenario from the list and **the trucks will execute a predefined set of actions**.

- The **STOP** button on the left is the biggest button in the Main vi. This works as a **panic button** that stops all trucks but also **closes all the connections correctly**. Try to use this button whenever you want to stop the program and **avoid the LabVIEW stop execution button**.

Before you press the LabVIEW run execution button, you have to **move the trucks by hand to strategic positions** . If the truck is set to start on the **Outer Lane**, you have to put the truck in the outer lane of the road network on the **opposite side of the bridge** with **anti-clockwise orientation**. If the truck is set to start on the **Inner Lane** you have to put the truck in the inner lane of the road network on the **opposite side of the bridge** with **clockwise orientation**. Finally, if the truck is set to start on the **Inner Road**, you have to put the truck in the inner road of the road network on the **opposite side of the bridge** with **clockwise orientation**. The truck is set to start at the waypoint number **1**. You can re-set this when fetching the trajectory waypoints (see Chapter 4)

Chapter 6

Getting Started With The Visualization Tool

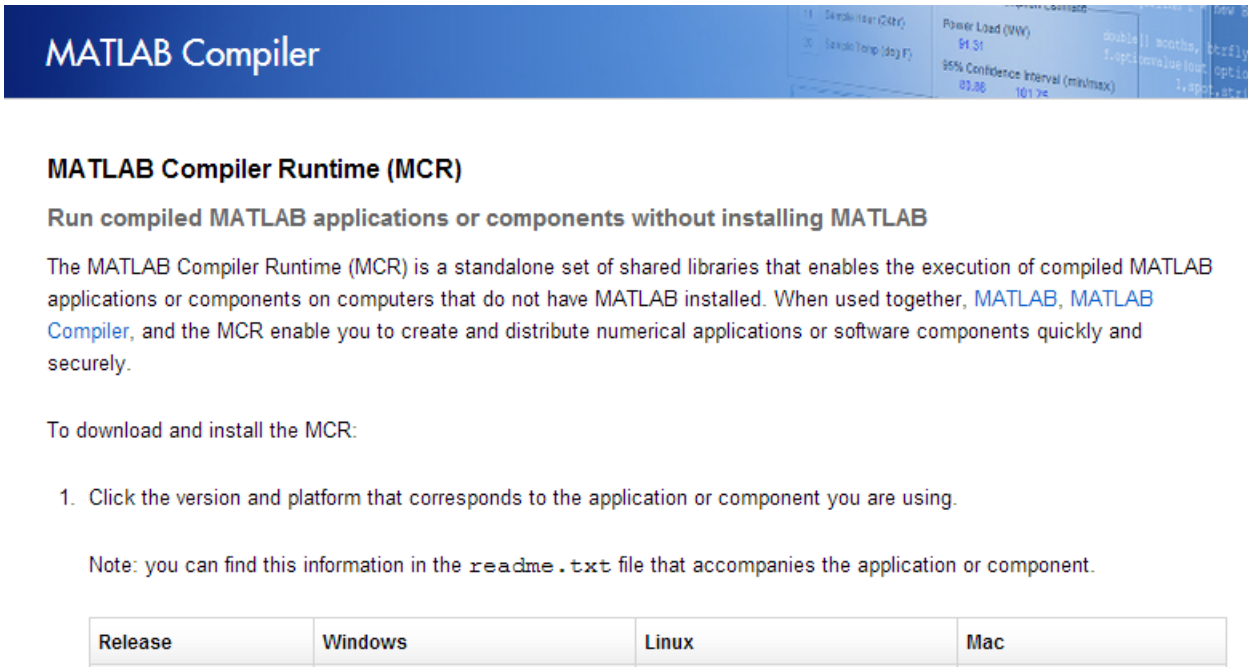
To help with the visualization of the project we created a tool that shows all the important data that the user might need to know in a more user friendly way. This is done by resorting to a communication **TCP/IP** between the computer that is running the **LabVIEW** software and any computer with access to an internet connection. Note that not all the data available is displayed here, but only the sufficient to aid the user understand what is happening at the moment. Also, the data of every run is saved in a **MATLAB** data file type, which will be helpful to posterior data analysis.

6.1 Prerequisites

- Once the [8] is already running you only need to run the [8]. Since this tool communicates with another computer via **TCP/IP**, an internet connection is obviously needed.

- Next, you need to have a **MATLAB** version installed in the computer where the **Visualization Tool** is to be run. In case that it is not possible there is an alternative that is to only install the **MATLAB Compiler Runtime (MCR)**[1]. After downloading the version that suits your needs, you simply need to run the **executable file** that we generated in case you use **Microsoft Windows**, or, if you use **Mac OSX**, you can run the **app** that we also generated.

[1] `http://www.mathworks.se/products/compiler/mcr/`



MATLAB Compiler

MATLAB Compiler Runtime (MCR)

Run compiled MATLAB applications or components without installing MATLAB

The MATLAB Compiler Runtime (MCR) is a standalone set of shared libraries that enables the execution of compiled MATLAB applications or components on computers that do not have MATLAB installed. When used together, MATLAB, MATLAB Compiler, and the MCR enable you to create and distribute numerical applications or software components quickly and securely.

To download and install the MCR:

1. Click the version and platform that corresponds to the application or component you are using.

   Note: you can find this information in the `readme.txt` file that accompanies the application or component.

Release	Windows	Linux	Mac
R2013a	32-bit / 64-bit	64-bit	Intel 64-bit
R2012b	32-bit / 64-bit	64-bit	Intel 64-bit
R2012a	32-bit / 64-bit	32-bit / 64-bit	Intel 64-bit

2. Save the MCR installer file on the computer on which you plan to run the application or component.
3. Double click the installer and follow the instructions in the installation wizard.

Figure 6.1: MATLAB Compiler Runtime (MCR) download.

- The computer that runs the **Visualization Tool** doesn't need any special graphical capabilities. Nonetheless, we recommend that you use a computer with a **considerable processing capability**. This is a requirement because the tool **receives new data** every $100ms$. If the computer isn't capable of processing the data and display it in that time window, at some point it will start processing data that is not updated. This can be seen by a **delay** between what you see in our tool and what is happening in **real-time**.

6.2 Running The Visualization Tool

To run the **Visualization Tool**, as mentioned in Section 6.1, you can run the **executable file** (or the app file in case of **Mac OSX**) or, if you prefer to run it within **MATLAB**, simply run the *SML.m* file. In both cases, the first window to appear is the one in Figure 6.2.

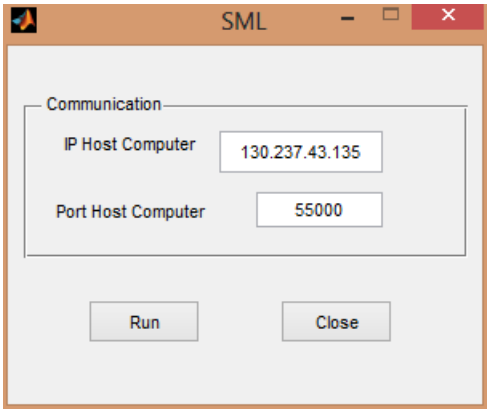


Figure 6.2: Connection settings.

Here you have to define the settings for the **TCP/IP** connection between your computer and the computer running the **LabVIEW** program. You have to set both the **IP address** and the **port number** of the remote computer. If you don't know the IP address of the remote computer just type '**ipconfig**' in the command line and the IP address of that computer will be shown. The port number should preferably be a number between $49152 - 65535$, but be aware that it must be the same in both sides. So because the default value in the **LabVIEW** program is 55000, we recommend to use it as port number. If by any chance you want to change it don't forget to change in both places. The values you see in Figure 6.2 are the **default values**, only because it is the current configuration in the **Smart Mobility Labs**' computers.

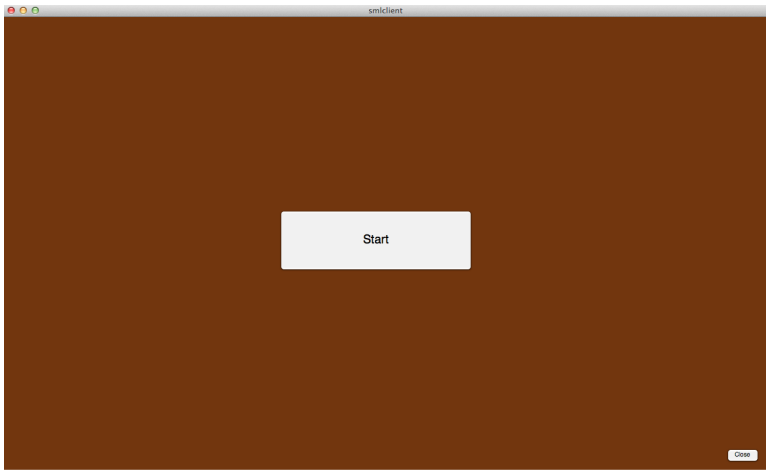


Figure 6.3: Start screen of the visualization tool.

After you push the run button your connection settings are defined and a start screen should appear. Preferably, you should push the start button when you have done all the required steps in the **LabVIEW**

program-this means that at this point the **LabVIEW** program should be waiting for an order to proceed. If you press it before time, it can cause some connection issues because there is a timeout in the connection. Summing-up: make sure to run the **LabVIEW** program first with the **Visualization Tool** option turned on.

6.3 Visualization Tool

Next, we have two examples of our **Visualization Tool**, each one representing the **layout** that we decided that better suited our scenarios. Of course you are free to improve it and adjust it to your needs. Both versions use the same base but one is more concerned in fuel related data and the other more concerned with the traffic lights status. We will briefly explain what each one of the **panels** represents.

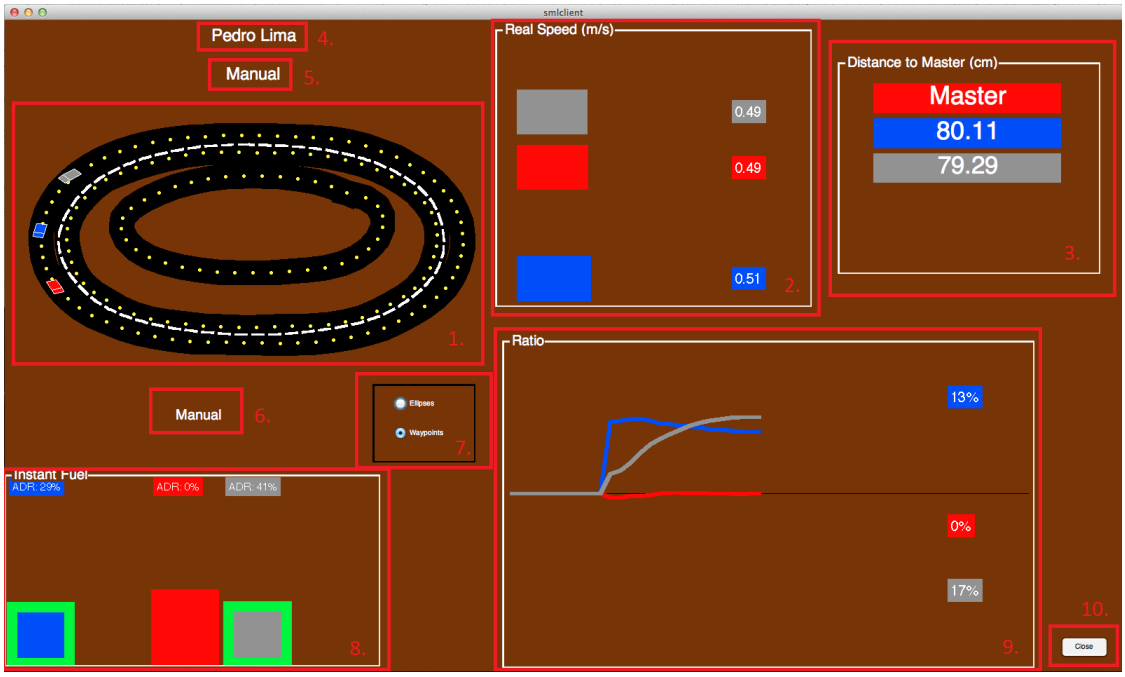


Figure 6.4: Visualization Tool (Pedro Lima).

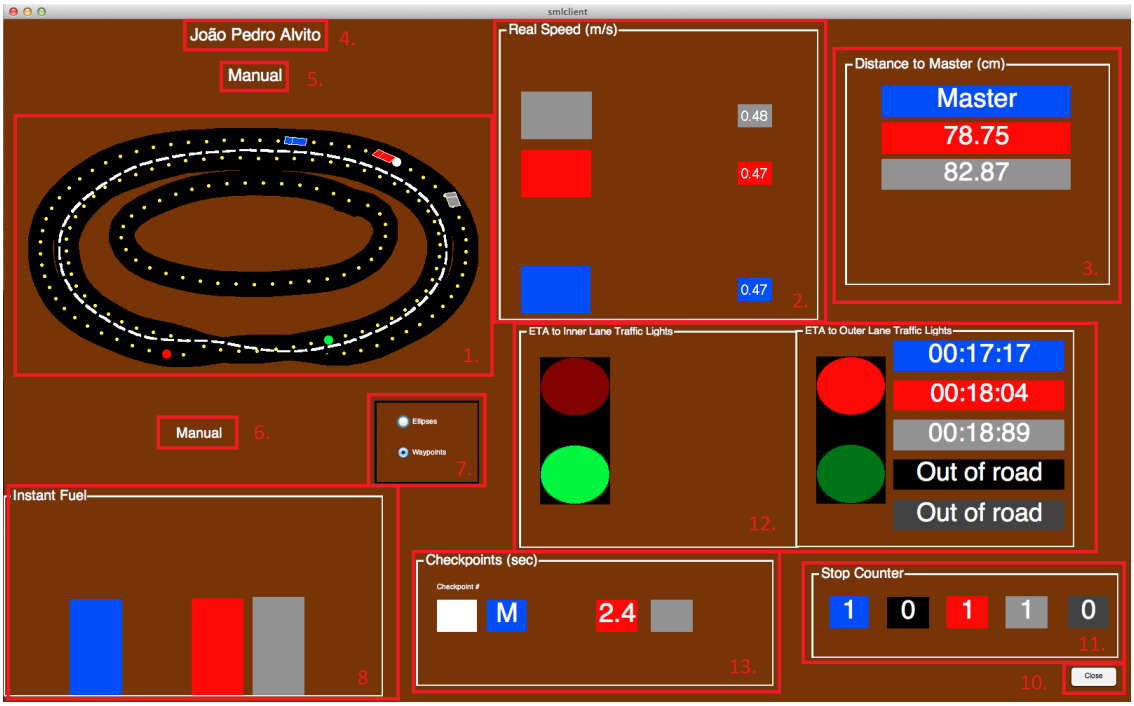


Figure 6.5: Visualization Tool (João Pedro Alvito).

- **1.** This is, most likely, our most important block, mainly because it's the one that contains more information. This is basically a transcription of the reality to our display. In it, you can see the current position of each truck that is visible in the **Qualisys** system, through the small rectangular shaped trucks colored accordingly to the real trucks' colors. It's also possible to see how the **road network** looks. The **Visualization Tool** simply receives the waypoints and processes it in order to display the several roads and its lanes in black, and also the **white dashed line** that we usually see in real life. In the lanes, it is possible to display the waypoints, representing them as **yellow dots**. Additionally, in the case of Figure 6.5, there are extra colored dots: the **white dot** represents a checkpoint that is used to measure the time distance between each consecutive truck (it is possible to have several checkpoints); there are **two dots** that can be either **red or green**, that at the same time give the **position** of the traffic lights of both lanes and its **status** (either if the traffic light is green or red);

- **2.** This block shows the real speeds of the trucks, *i.e.* in SI units *(m/s)*. To make it easier to distinguish between trucks, we used a color code that translates the user to the real color of the trucks that are being used. There are two fields where it is possible to get information about the speeds: a horizontal bar that is helpful to analyze the **relative speed** between trucks; a text field with the the **absolute speed** value; Note that this block only displays the speed information of the trucks that are being used, so if for example a truck is not being used or even if it is inside the visible zone but is out of the road, its speed information will be suppressed from the display;

- **3.** This block shows the most important information about the platooning that is the **Platooning Distance**. This measurement represents the size of the gap between each consecutive truck that is part of the platoon. For better understanding, we are using the measurement in *cm* and it is the real distance between the trucks and not the **real world** distances. Note that the first truck of the platoon, **the master**, naturally doesn't have any distance to display so it just identifies itself as the **master**. To improve the visualization, the list of the **Platooning Distance's** order reorders itself according for the order in the platoon; this way the information about the platoon order is embedded with the **Platooning Distance**. If a truck isn't part of the platoon the **Platooning Distance** is obviously suppressed from the display;

- **4.** This field simply shows what is the **Platooning Distance** of the current scenario;

- **5.** Here you can see what is the **name** of the current scenario. The names are pretty self-explanatory because this way it is easily understood what the scenario is meant for;

- **6.** This shows a simple explanation of what is happening at each moment. It is an one line sentence that illustrates what might not be obvious; for example, it says what a truck is trying to accomplish or if some event related to a traffic light is taking place;

- **7.** This block contains the optional choices that the user can do. The **"waypoints"** option is simply a radio button that turns "on" or "off" the waypoints display in **1.**. This option is defaulted to be "on" because we believe that the waypoints appearing are a great aid to observe the trucks' trajectory.

The other option that we have available is **"ellipses"**. We provide this option because the graphical roads in **1.** are dynamically generated, so sometimes, depending on the trajectory generated, the roads can be displayed in a rather strange way. Since the roads that we created in the **Smart Mobility Lab** are shaped like ellipses we tried to process the road data that we received from the trajectory generation and adjust it to ellipses. Sometimes, this option improves the display of the roads but of course this is not a viable option if the road is not shaped like an ellipse so feel free to try this option and see if it improves the display of the **road network** that you are currently using;

- **8.** Here you can see a relative indication of the **instantaneous fuel consumption**. It allows you to better perceive how much fuel the trucks are consuming in relation to each other. In Figure 6.4 there are two extra pieces of information: the percentage of **Air Drag Reduction (ARD)** and, also surrounding the vertical bars of the instantaneous fuel, you can see either **red or green edges**. If that edge is green, it means that for that particular truck it is being beneficial in terms of fuel consumption to be part of the platoon, otherwise the edge will appear red. For more technical information refer to [4] and [8];

- **9.** Here you can see a **fuel ratio** for each truck involved in the scenario. This is the ratio between the accumulated fuel consumption, when the truck decides to do catch up, and the accumulated fuel consumption that the truck would have if it had continued alone. For more technical information refer to [8];

- **10.** This close button, when pushed, closes the program. We do not recommend to use this button as a first choice, it is preferable to use the stop button in the **LabVIEW** program which you can see in Figure 5.5. This way the **LabVIEW** program sends a closing order to the Visualization Tool which allows it to close by itself and then closing all the **TCP/IP** connections from both sides;

- **11.** This blocks provides a stop counter that shows the number of times each truck has stopped. This is helpful to see if a truck stopped in a traffic light and how many times it did;

- **12.** This block has two different sub-blocks that are basically the same but the information is divided in inner and outer lane. So on the left side we have the traffic light information about the inner lane and on the right side there appears the same information for the outer lane. These sub-blocks have all the information about the **traffic lights** apart from its position: the status of the **traffic light**, either if it is red or green; the **estimated time of arrival (ETA)** of each truck to the traffic light of the correspondent lane. The time appears in a digital form (minutes:seconds:centiseconds). To improve the display we decided to order the **ETAs**, such that the closest truck to the **traffic light** will always be on the top of the list;

- **13.** Here you can check which are the time distances at a given **checkpoint** (time distance is given in seconds). Note that this feature is only valuable if there is a platoon. If a truck is the master, the letter **'M'** will appear in the same place where it would show the time. Once the master passes through the checkpoint, as showed in point **1.**, all the other fields of the trucks that are in the platoon are erased and updated with the time distance as soon as the respective truck goes trough

the same **checkpoint**. You can use several **checkpoints**, but for bigger **Platooning Distances** we recommend to use only one checkpoint. Don't forget that the only values that are shown are the ones from trucks in the platoon.

Chapter 7

Troubleshooting

7.1 Trucks

If you have read all the manual and you the trucks are behaving as they supposed to, please go through the following checklist:

- Check if the batteries are **charged**;
- Check if all the **Polulu boards** have a **yellow** led **on** (see Chapter 2);
- Check for any **"Note: write failed"** or **"unix_error"** in the **Cygwin terminal** (see Chapter 2);
- Check all the **ID**, **Port** and **QTM ID** in the panel **4.** (see Section 5.3);
- Check the **IP address of the QTM computer**;
- Check if the value sent for **Speed** and **Steering** in bits present in panel **1.** are different from **127** (see Section 5.3);
- Check if the **PC's T-Motes are blinking**, meaning that they are communicating (see Chapter 2);
- Check if the **trucks' T-Motes are blincking** as well (see Chapter 2);
- Check if the truck motors are "**On**" (see Chapter 2);
- Check if the trucks are **being recognized by the MoCap** (see Chapter 3);
- Check if the trucks are **initially positioned** as they should (see Section 5.3);
- Check if there's **any commented block**.

7.2 Trajectory Creation

One problem that might arise in the trajectory creation, is a message saying **"No Welcome Message Received"**. This happens when the program is trying to connect to the **Qualysis** system and it receives an error probably due to a last connection that wasn't closed. The most effective solution that we found was restating **MATLAB**.

7.3 Visualization Tool

When using the **Visualization Tool** sometimes you might get some error when receiving the data from the road network in MATLAB. We are not completely sure about the source of the problem; nevertheless, this issue can be solved with a simple restart of the **Visualization Tool**.

You still can't get everything to work? Please, don't hesitate to **contact us** (`pfrdal@kth.se` and `jpfa@kth.se`).

Bibliography

[1] Qualysis AB. *QTM Getting Started*, 2011.

[2] Qualysis AB. *QUALISYS MATLAB CLIENT v1.8*, 2011.

[3] Qualysis AB. *QTM User Manual*, 2011.

[4] J. P. Alvito. Implementation of traffic control with heavy duty vehicle anti-platooning. Master's thesis, KTH The Royal Institute of Technology, 2013.

[5] M. Amoozadeh. *Smart Mobility Lab Manual*, 2013.

[6] A. Hauksson, B. Mengana, C. Westermark, F. Svensson, J. Lycke, J. Sundberg, K. Imhäuser, and S. van de Hoef. Final report in automatic control project course. Technical report, KTH The Royal Institute of Technology, December 2012.

[7] A. Hernandéz and D. Huertas Péres. Communication between pc and motes is tiny os. Technical report, KTH The Royal Institute of Technology, may 2012.

[8] P. Lima. Implementation and analysis of platoon catch up scenarios for heavy duty vehicles. Master's thesis, KTH The Royal Institute of Technology, 2013.